Lecture 13

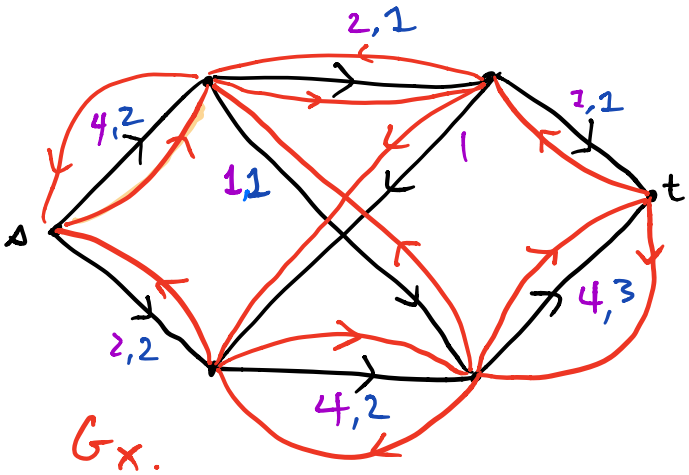
Plan:

1) algorithm for max flow
2) general min cut.

Recap: Augmenting flows:

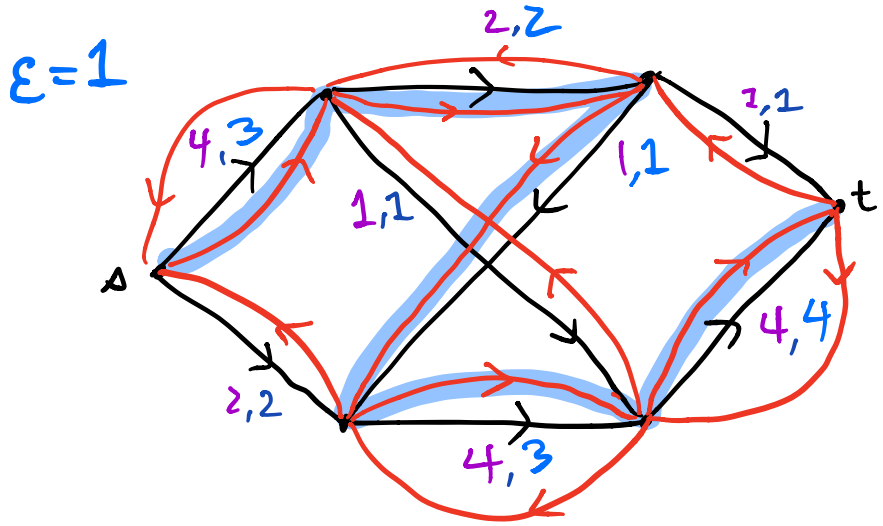
- Given a flow x, compute residual graph G_x.

e.g. $\ell = 0$, u, x



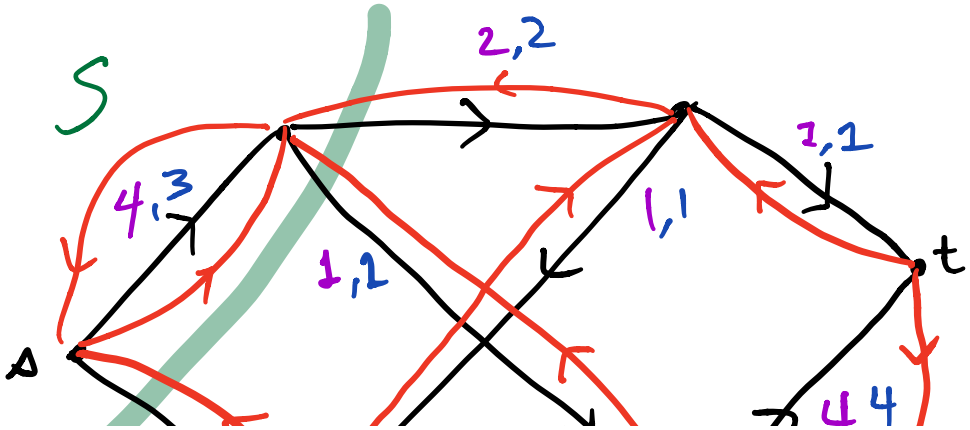
G_x.

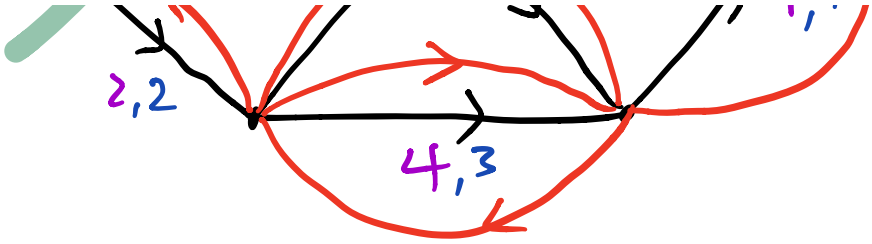
- Clf $\exists$ s-t path in residual, ε - more units of flow can be sent along it.



- if no s-t path, is cut S with capacity

  $C(S) = |X|$, terminate.





Problem: might not terminate!

- if irrational may run forever.
- if rational, can multiply capacities by s.g. to make integers.
- If capacities integral, can take $\epsilon > 0$ to be integral, so must terminate.

- in integral case, # steps

- naively bounded by

$$\sum_{e \in t} |u(e) - \ell(e)|.$$

but this is not polynomial in input size!

Remember: need only $1 + \log_2 |\ell(e)|$ bits to represent $\ell(e)$.

remedy:

Edmonds-Karp alg.

- variant of aug. flows.

- terminates in poly(m, n) steps, $m = |E|$, $n = |V|$, regardless of the capacities

"Strongly polynomial time".

- Works even if capacities irrational.
- Unknown if ∃ strongly polynomial time for general LP's.
- **Algorithm**: same as before, but use shortest s-t path in residual.

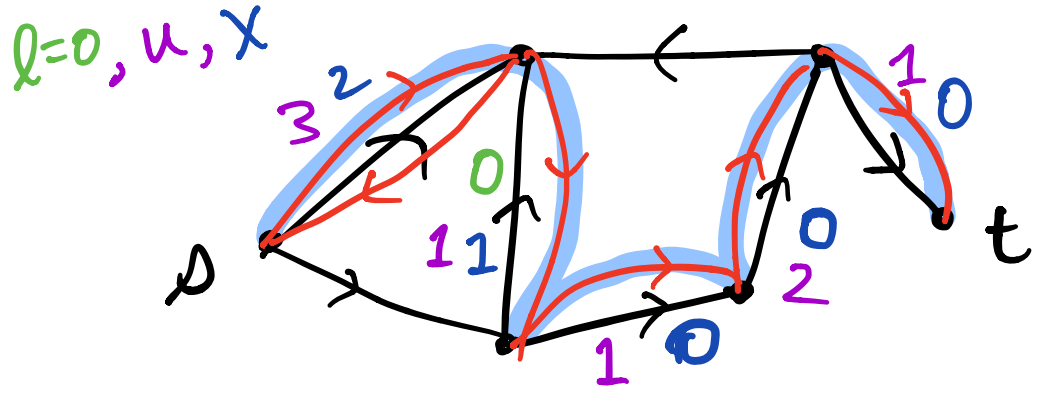
Analysis idea: show iterations increase s-t distance in residual.

- For $v \in V$, let $d_s(v)$ denote distance from s to v in G_x (length of shortest directed s-v path).
- Let P shortest s-t path in G_x.

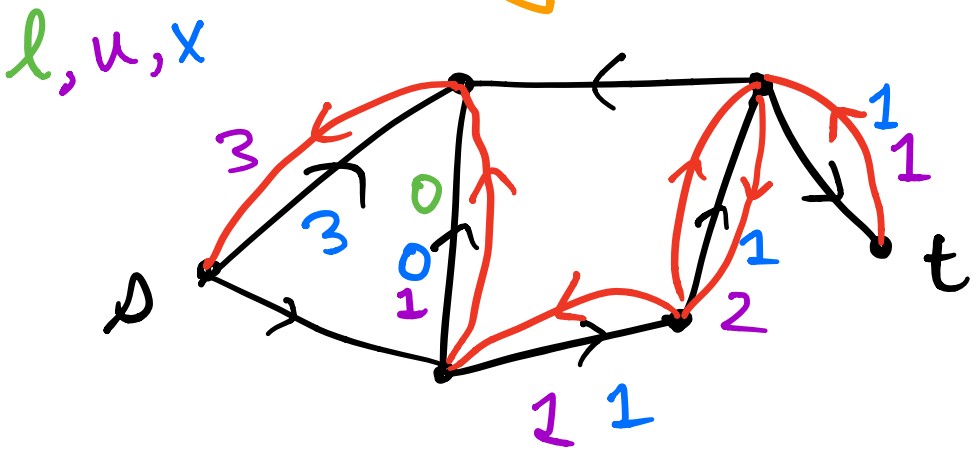
  $P = \underset{\| v_0}{s} - v_1 - v_2 \cdots - \underset{\| v_t}{t}$, $d_s(v_j) = j$.
- x' flow after augmenting along P (by as much as possible).
- Let d_s' be distance labels for $G_{x'}$.
- Note: any edge (i,j) added to G_x goes opposite direction of P.

$\cdot \quad \ell(i) - d_s(i) + 1$. Δ

i.e. $ds(i) - ds(j)$ ...



augment.



- After augmenting,

$$ds(j) - ds(i) \leq 1 \quad \star$$

for every edge (i,j) in $E_{x'}$.

— in G_x, not $G_{x'}$!

(because if $(i,j) \in E_x$, ✡ automatic.
if (i,j) new, $ds(i) - ds(j) = -1$
from Δ)

- for any $j \in V$, Sum ✡ along edges of shortest s-j path P' in $G_{x'}$,

$$ds(j) = \sum_{(i,j) \in P'} ds(j) - ds(i) \leq |P'| = ds'(j).$$

In particular,

$$ds(t) \leq ds'(t).$$

- Distance to t can

increase $\leq n-1$ times.

- But how often must it increase?

Each iteration, some edge with $ds(j) = ds(i) + 1$ is removed from G_x.

(b/c P shortest, some edge of P flips).

- Thus after $\leq m$ iterations must get $ds(t) < ds'(t)$.

(one inequality in telescoping sum becomes tight).

- **In summary:**

  (i) # augmentations $\leq m(n-1)$

  (ii) time to build G_x, find $P = O(m)$.

  $\Rightarrow$ running time $O(m^2 n)$. $\square$

  not tight.

The initial feasible flow

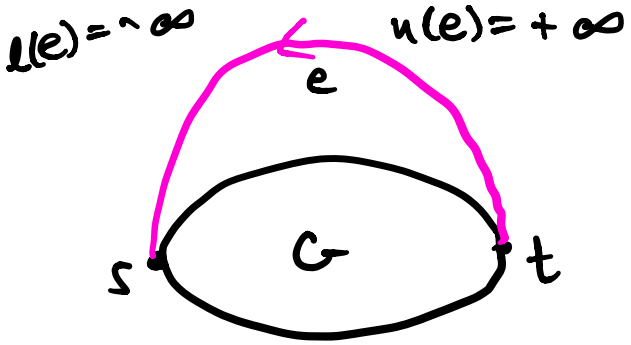
We still need to find flow to start with!

- if $l(e) \leq 0 \leq u(e)$ $\forall e$, use $x = 0$.
- if not, feasible flow is maxflow for another network that's easy to initialize.

Circulations

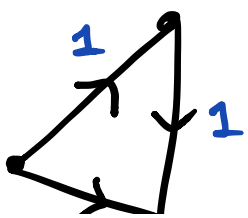
- first reduce finding feasible flow to finding "circulation" in new graph $\tilde{G}$:

$\tilde{G} =$

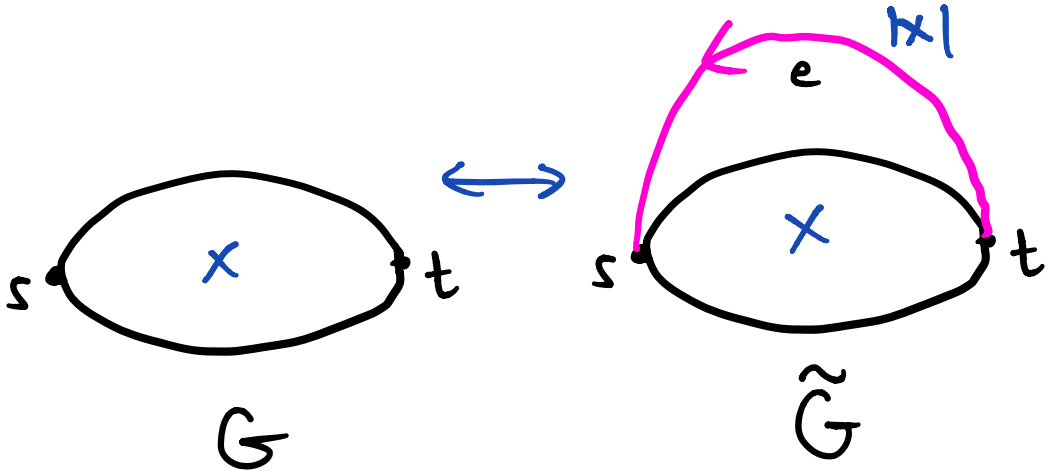


- Define **circulation** of G, u, ℓ as flow satisfying conservation at **all** $v \in V$ (s, t no longer special).

e.g.



- Bijection between flows in G & circulations in $\tilde{G}$:



(just add $|x|$ to new edge).

finding Circulations!

Let $G=(V,E)$ arbitrary digraph w/ capacities ℓ, u.

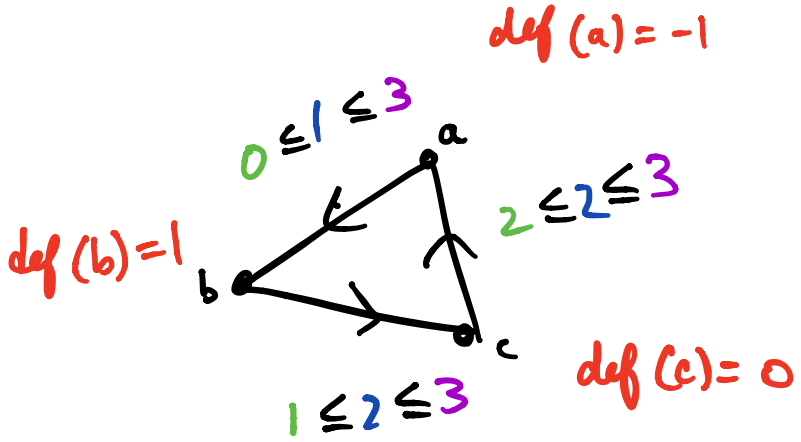
- First, choose arbitrary y_e with

w.r.t. $\ell(e) \le y_e \le u(e)$.

- y_e need not be flow; define **deficit** at v

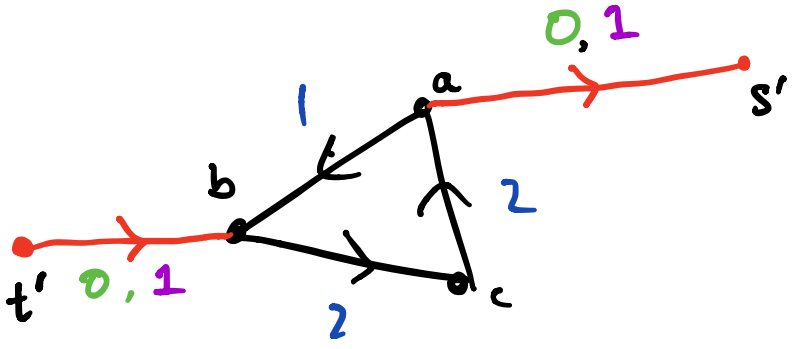
$$\text{def}(v) := \sum_{\delta^+(v)} y_e - \sum_{\delta^-(v)} y_e$$

e.g.



- To fix: add extra vertices/edges to "supply" deficit

e.g.



Formally: Let $G' = (V', E')$ with $V' = V \cup \{s', t'\}$.

(i) add two vertices s', t'

(ii) Let $V^+ = \{v : \text{def}(v) > 0\}$.

$V^- = \{v : \text{def}(v) < 0\}$.

(iii) For $v \in V^+$, add edge $e = (v, t')$ with $\ell(e) = 0$, $u(e) = \text{def}(v)$.

For $v \in V^-$, add $e = (v, s')$ w/ $\ell(e) = 0$, $u(e) = -\text{def}(v)$.

- Setting flow on new edges equal to upper capacities gives feasible flow for network G' with source s', sink t'.

- initial value is

$$\sum_{v \in V^-} e(v) < 0.$$

- Using this initial flow, apply Edmonds-Karp to find max flow X;

note $|X| \leq 0$.

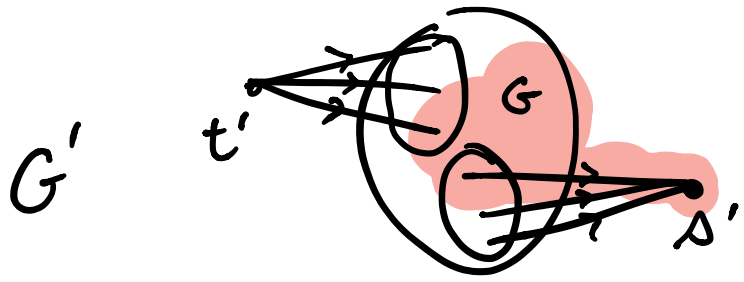
- If $|X| = 0$, restricting X to E gives circulation.

- If $|X| < 0$, then no circ. exists because circulatin gives flow of value 0.

- **In summary:** to find feas. flow in G, find circulation in $\tilde{G}$ by solving max flow in $\tilde{G}'$.

When is a flow network feasible?

- Enough to decide if there's a circulation.
- Use max-flow min-cut in G'.



- s'-t' cut in G' is $S \cup \{s'\}$, $S \subseteq V$.

- Capacity is

$$c_{G'}(S \cup \{s'\}) = c_G(S) = \sum_{\delta^+(S)} u(e) - \sum_{\delta^-(S)} \ell(e).$$

↑ (b/c $\delta(S)$ not in cut, $\delta(t)$ have 0 lower capacity).

To summarize:

Theorem G, ℓ, u admits circulation iff $\forall S \subseteq V$,

$$\sum_{\delta^+(S)} u(e) - \sum_{\delta^-(S)} \ell(e) \geq 0.$$

Corollary Flow network feasible iff $\ell(e) \leq u(e) \; \forall e$ and

$$\sum_{\delta^+(S)} u(e) - \sum_{\delta^-(S)} \ell(e) \geq 0$$

$\forall S$ s.t. $|S \cap \{s,t\}| \neq 1$.

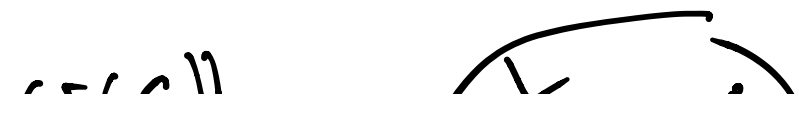
(General) min cut.

- Assume now $\ell = 0$, so cut capacity is just

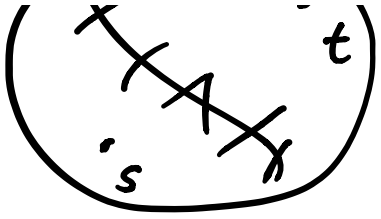
$$u(\delta^+(S)) := \sum_{\delta^+(S)} u(e).$$

- We've shown how to find min s-t cut using max-flow.

- Can also solve



$$\min_{s\text{-}t \text{ cuts } S} u(\delta(S))$$



in undirected graph by

- What about **global** minimum cut (not for fixed s, t)

- Can reduce to $2(n-1)$ maxflows:
  - (i) choose arbitrary vertex s
  - (ii) for any $t \in V \setminus \{s\}$, solve for min s-t cut, min t-s cut, [illegible] smaller.

take whichever s...
Do this for all t.

- Fastest maxflow algs take around $O(mn)$ time (Goldberg-Tarjan), so our naive alg. takes $O(mn^2)$ time.
- Has-Orlin used relationships between the $O(n)$ flow problems to give an $O(mn \log(n^2/m))$ time alg for global mincut.

Today: different algorithm, not using flows, with comparable runtime to Hao-Orlin.

uses property of "diminishing returns" aka submodularity

Def: For $A, B \subseteq V$, define

$$u(A:B) = \sum_{\substack{i \in A, \\ j \in B}} u((i,j))$$

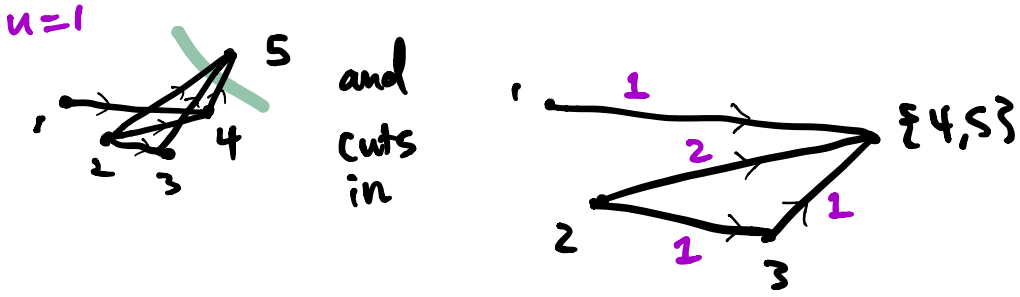
* WLOG assume complete

Algorithm idea:

- starting with vertex, build "max adjacency ordering",

i.e. greedily add vertex w/ least cost to previous ones.

- Consider cut from last vertex, and also cuts obtained by shrinking last two vertices.



(when shrinking creates duplicate edges, weights add).

- Claim: best cut considered this way is global min cut.

Formally:

Algorithm (Stoer-Wagner)

MINCUT(G) # outputs cut.

- Let v_1 any vertex of G
- $n := |V(G)|$

create ordering

- initialize $S = \{v_1\}$
- for $i = 2 \dots n$:
  - $v_i = \arg\min_{v \in V \setminus S} w(S : \{v\})$.
  - $S \leftarrow S \cup \{v_i\}$
- if $n = 2$:
  - return $\delta(\{v_n\})$.
- else:
  - Get G' by shrinking $\{v_{n-1}, v_n\}$

recursive call

▷ Let $C = \text{MINCUT}(G')$

▷ return less costly of

$C, \delta(\{v_n\})$.

Analysis: uses a claim.

Claim: $\{v_n\}$ is a min v_{n-1}-v_n cut.

Claim ⇒ correctness:

- The min cut is either a min v_{n-1}-v_n cut, or not.
- If it is, claim ⇒ alg. outputs it. ✓
- If not, by induction on $n = |V(G)|$, algorithm outputs min cut in G'. ✓.

(b/c min cut does not split v_{n-1}, v_n, min cut in shrunk G' is same).

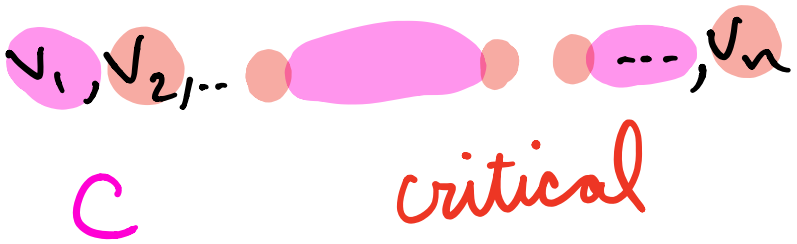
Proof of Claim:

Let $v_1, v_2, \ldots, v_{n-1}, v_n$ be the ordering from alg.

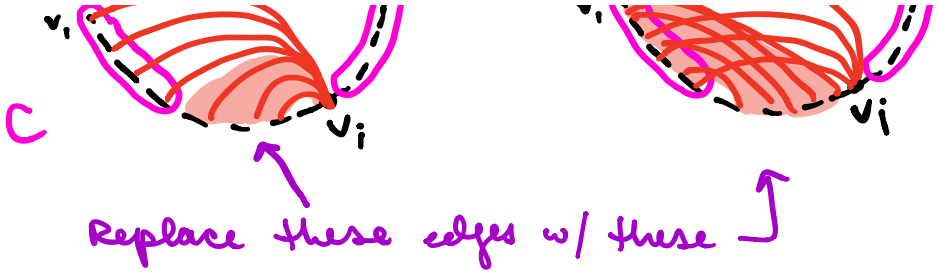
- $A_i :=$ sequence $v_1 \ldots v_{i-1}$
- Consider candidate cut, i.e. $C \subseteq V$ s.t. $v_{n-1} \in C, v_{n-1} \notin C$.
- Want to show $u(\delta(A_n)) \leq u(\delta(C))$,

i.e. cut from $\{v_n\}$ is better than C.

- define v_i to be **critical** if either v_i or v_{i-1} in C but not both.



- **Subclaim:** Define $C_i := A_{i+1} \cap C$ If v_i critical, then $u(A_i : \{v_i\}) \leq u(C_i : A_{i+1} \setminus C_i)$.



Subclaim suffices, because

subclaim $\Rightarrow$ $u(\delta(A_n)) \leq u(\delta(C))$

$u(A_n : v_n)$ $\quad$ $u(C_n : A_{n+1} \setminus C_n)$.

because v_n is critical.

proof of subclaim:

by induction on seq. of critical vertices.

- (base:) true for first critical vertex. (see picture.)
- (inductive:) Assume true for critical v_i, let v_j next critical.
- Then

$$u(A_{i+1}) + u(A_i \setminus A_{i+1} : \{v_i\})$$

$$u(A_i : \{v_j\}) = u(A_i : \{v_j\}) \cdots$$

$$\leq u(A_i : \{v_j\}) + u(A_j \setminus A_i : \{v_j\})$$

by ordering.

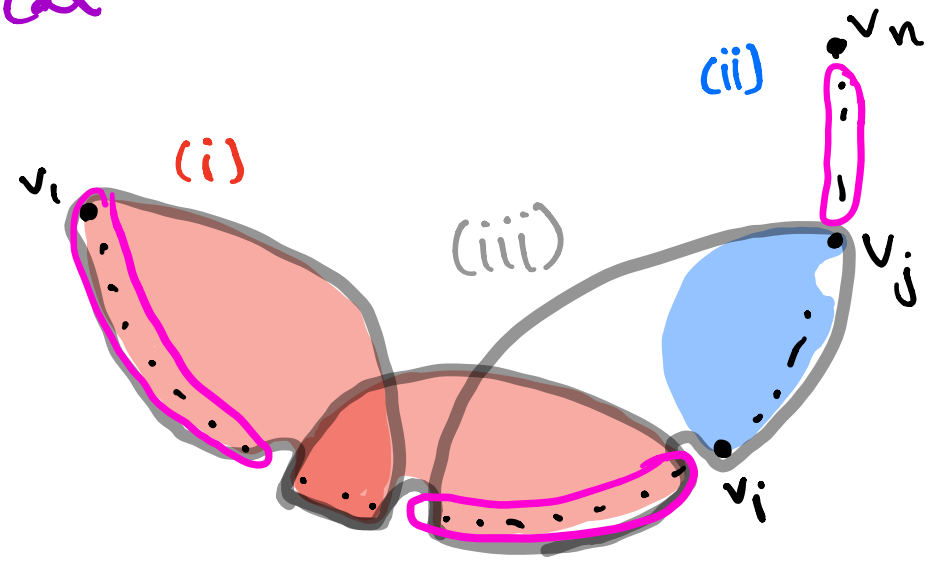
$$\leq u(C_i : A_{i+1} \setminus C_i) \quad \text{(i)}$$

induction

$$+ u(A_j \setminus A_i : \{v_j\}) \quad \text{(ii)}$$

$$\leq u(C_j : A_{j+1} \setminus C_j). \quad \text{(iii)}$$

v_j is next critical



□

Running time:

Depends how you do recursion.

- Fibonacci heaps ⇝ each iteration takes $O(m + n \log n)$ time
- overall, $O(mn + n^2 \log n)$ time.

Submodularity

- Hidden property of $f: S \mapsto u(\delta(S))$ made the algorithm work.
- function $f: 2^V \to \mathbb{R}$ **submodular** if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.

- **Examples:**
  - $f(S) = |S|$,
  - $f(S) = u(\delta(S))$ for u nonnegative, G undirected

Submodularity equivalent to "diminishing marginal returns":

For $S \supseteq T$, $v \notin S$,

$$f(S \cup \{v\}) - f(S) \leq f(T \cup \{v\}) - f(\{v\}).$$

- Above algorithm can be extended to minimize any symmetric ($f(S) = f(V \setminus S)$) submodular function.